SMS Spam Detection Project Report

Erik Peterson

Abstract—SMS (text) messages, due to their prolific use, are a rich breeding ground for spam. The small size of these messages can make spam detection difficult, and the instant nature of them means that long-running spam-catching algorithms are not viable. Machine learning offers a way to filter most spam messages while at the same time stopping very few legitimate messages. This paper examines several algorithms that provide varying degrees of spam protection and suggests a combination of algorithms to use in a spam filtering environment.

1 Introduction

SMS messages, with their wide use (even wider when you add other instant messaging solutions) are a ripe breeding ground for spam. The short length of these messages, as well as the huge array of acronyms and other means of shortening messages it spawned, can make it difficult to tell what is a legitimate message and what represents spam.

Almeida and Hildago created a corpus of SMS messages, including markings, in an effort to provide a useful set of test data to explore message filtering. This corpus contains 5,574 messages, of which 747 are spam. These messages are at [2]. They also authored a paper that details their initial efforts towards filtering spam messages. [1]

This paper extends the previous work by applying different algorithms. It also discusses the steps taken to prepare SMS messages for the machine learning algorithms. The goal is three-fold:

1) Block as much spam as possible
2) Block few (ideally, no) legitimate messages
3) Apply a model quickly (to minimize any delay in delivering the messages)

The rest of this paper is structured as follows. Section 2 deals the testing setup, including the RapidMiner software used to run the algorithms. Section 3 details the preprocessing necessary to use the text messages in a machine learning environment, as well as the means attempted to turn the text into numerical data in RapidMiner. Section 4 discusses the algorithms applied, detailing three algorithms that showed the best results. Section 5 deals with the results of the machine learning algorithms. Section 6 recommends a combination of machine learning algorithms to create a good filtering setup. Finally, Section 7 presents conclusions.

2 Testing Setup

The test machine is a Lenovo Flex 14" laptop running Windows 10. The laptop features an 8th-generation Intel Core i5 processor (4 cores/8 threads), 8 GB of RAM, and a 256 GB SSD.

Much of the text preprocessing was done as a Java program created using Android Studio (as a standalone Java process rather than an Android app—I already had Android Studio installed, and it made little sense to install a separate IDE for this project).

The tokenization, further preprocessing, and the machine learning algorithms were run in RapidMiner Free Edition, version 9.0.003. As this was my first real experience with RapidMiner, I used [4] [5] [6], and [7] to help me get familiar with RapidMiner, and in particular text processing in RapidMiner.

As done in the paper accompanying the SMS corpus, the data is split to create a training set (30% of messages) and a test set (the remaining 70% of the messages). [1]

The Java project, the resulting preprocessed text, and the detailed results of the algorithm runs is available at https://github.com/ErikAggie/SMSSpam.

3 Text Message Preprocessing

Preparing the provided text messages for machine learning was done in two steps: custom text cleanup and text preprocessing in RapidMiner.

3.1 Code-based Cleanup

The SMS message corpus [2] present some issues that, if left unresolved, could bias the machine learning algorithms. Many of these issues seem to be the result of a text encoding error: either the corpus was not saved properly or else Windows did not properly understand it.

To correct these issues, the Java program makes the following changes to the messages:

- Removes any junk characters that appear at the beginning of a line.
- Ensures that each line in the file has a tab between the label and the messages (some had a space instead).
- Replaces some HTML strings with the proper character (<, >, and &).
- Corrects a number of other characters/character combinations that are incorrect, likely the result of a file encoding issue. In some cases the actual character could not be determined; these characters are simply removed.
- Runs a check to see if any special characters remain.

Many numbers (likely phone numbers, for the most part) are not present in the SMS corpus, presumably for privacy reasons. This was left untouched. In theory such data could improve the performance of spam catching (i.e. if a specific number is in many spam messages, that would be a good

TABLE 1
Preprocessing Performance with Perceptron

Preprocessing applied	Ham Blocked	Accuracy
None	0.24%	98.00%
Lowercase all	0.21%	97.71%
+filter stopwords, Stem (Porter)	0.80%	96.97%
+n-gram generation (3)	0.62%	97.31%

token to use for discovering further spam messages), but for privacy reasons it might be wise to perform the same generalization when building and applying production models.

3.2 RapidMiner Text Preprocessing

The data created by the Java program was imported into RapidMiner as training and test datasets. Almeida, Hidalgo, and Yamakami proposed two sets of characters for tokenizing the data, which are replicated in this paper. The first (tok1) is space plus period, comma, and colon; the second (tok2) adds the dash character. [1]

Almeida, Hidalgo, and Yamakami did not attempt to process the data further, pointing to research into email spam that suggested that further processing tended to hurt spam filtering. [1] To check that claim I ran extra tests using three text processing options layered on top of each other: converting messages to lowercase, removing stopwords, and creating n-grams (three).

As Almeida, Hidalgo, and Yamakami expected, [1] there is a general degradation of performance as more processing is applied to the tokens, but the degredation is not uniform. kNN, for example, saw an uptick in performance when using tok1 and making all messages lowercase, but this was an exception. Perceptron (using token 1), shown in Table 1, is a typical case. ("Ham" in this and the other tables refers to legitimate text messages.)

In addition, while making all text lowercase, filtering stopwords, and stemming can reduce the number of tokens found, adding the n-grams added many more tokens, which severely slowed down the machine learning algorithms.

4 Algorithm Selection

In an attempt to cover a breadth of options, eight algorithms were applied and optimized. These algorithms fall into two basic categories:

- Algorithms that run quickly (Decision Stump, Deep Learning, kNN, Naive Bayes, Perceptron, and SVM). These require minutes, at most, to create and apply a model across the entire corpus.
- Algorithms that run slowly (LDA and Neural Network). These algoritms run slowly enough—LDA, unaided, takes well over an hour to apply its model—that some form of dimensionality reduction is warranted.

The "slow" algorithms present a problem with a time-critical application like text messaging. LDA, in particular, is very slow at applying a model, taking over an hour on my test machine. If LDA or Neural Network is to be viable in this situation, some form of dimensionality reduction is needed. (It should also be noted that the model application could well happen on a mobile device; Apple, for one, is known for running machine learning on iPhones for privacy reasons. In such cases algorithm speed is doubly important to save the user's battery.)

TABLE 2
Best Performance of Each Algorithm

Algorithm	Spam caught	Ham blocked	Accuracy
Decision Stump: tok1	18.07%	0.47%	88.90%
Decision Stump: tok2	18.07%	0.47%	88.90%
Deep Learning: tok1	89.98%	0.12%	98.59%
Deep Learning: tok2	**90.37%**	0.12%	**98.64%**
kNN: tok1	78.98%	0.32%	96.98%
kNN: tok2	76.82%	0.47%	96.57%
LDA: tok1	76.23%	0.03%	96.87%
LDA: tok2	77.80%	0.03%	97.08%
Naive Bayes: tok1	88.41%	4.69%	94.41%
Naive Bayes: tok2	88.02%	4.69%	94.36%
Neural Network: tok1	88.80%	0.27%	98.30%
Neural Network: tok2	87.82%	0.21%	98.23%
Perceptron: tok1	86.25%	0.24%	98.00%
Perceptron: tok2	87.23%	0.29%	98.08%
SVM: tok1	72.10%	0.00%	96.36%
SVM: tok2	72.89%	**0.00%**	96.46%

The issue, however, is that dimensionality reduction itself can take a great deal of time. PCA, which provides the best results for both LDA and Neural Network, is also slow to run. To combat this I used a few weight-based token selectors, which ran much faster but, in general, provided worse results.

5 Results

5.1 Overall Results

The best results from each of the algorithms across both tokens is shown in Table 2, with the best results in each measurement bolded. The optimizations made to the three best-performing algorithms are discussed in the following sub-sections.

Whereas the SMS paper deals with overall performance (that is, how well each algorithm finds both spam and legitimate messages) [1], my focus here is more nuanced. When filtering spam, it is of prime importance to block as few legitimate messages as possible. Failing to do so will result in frustrated customers wondering why some messages never get delivered. For this reason, the "Ham blocked" measure is key. In selecting the "best" optimization of an algorithms, blocking few true messages is given high weight, to the extent that, at times, a lower overall accuracy is accepted in order to minimize the blocking of legitimate text messages.

In all the cases discussed below, token 2 proved the best way to split messages.

5.2 Deep Learning

Deep Learning was the best performing algorithm overall, with scores that best the algorithms tried in the SMS paper (which did not run this algorithm) [1].

TABLE 3
Deep Learning Activation Function Options

Option	Spam caught	Ham blocked	Accuracy
Rectifier	83.10%	1.36%	96.61%
Tanh	90.37%	0.12%	98.64%
Maxout	58.35%	0.15%	94.43%
ExpRectifier	93.32%	1.89%	97.48%

TABLE 4
Deep Learning Hidden Layer Variations

Layers	Spam Caught	Ham Blocked	Accuracy
50 (one layer)	91.94%	1.71%	97.46%
50, 20	90.18%	0.59%	98.21%
50, 50	90.37%	**0.12**%	94.50%
100, 100	89.59%	0.35%	98.34%
100, 50	91.55%	0.56%	98.41%
60, 50	89.98%	0.21%	98.51%
50, 50	94.50%	1.71%	97.80%

TABLE 5
Neural Network Preprocessing Options

Preprocessing (Limit)	Spam Caught	Ham Blocked	Accuracy
None	70.73%	1.30%	95.05%
Info Gain (.008)	75.64%	2.74%	94.44 %
Info Gain Ratio (.03)*	0.00%	0.00%	86.95%
SVM (.008)	58.15%	0.86%	93.79%
PCA, (.95)**	87.82%	0.21%	98.23%

* Weights threshold higher as the lowest weight generated was .018
** Done with 5 nodes in the hidden layer; the others used 10 nodes

Deep Learning is very sensitive to the activation function. The default, Rectifier, was considerably worse than Tanh. Using the default setting of two layers of 50 nodes each, the available activation functions produced the results in Table 3.

Deep Learning also provides the ability to include dropouts in order to reduce the size of the network (and, I believe, to reduce overfitting). This proved ineffective when tried.

In addition, RapidMiner lets you tune the size of the hidden layers. Experiments with a variety of options suggest that that default of two layers of 50 nodes each gives the best results. The variations attempted are shown in Table 4.

5.3 Neural Network

Neural Network proves to be a very good classifier if provided with the right dimensionality reduction, with results falling just short of Deep Learning. It is also, next to LDA, the longest-running algorithm with no dimensionality reduction done beforehand.

In an effort to improve Neural Network's performance I tried selecting tokens using several weight-based methods:

- Weights by Information Gain
- Weights by Information Gain Ratio
- Weights from SVM

TABLE 6
SVM Options

Kernel	C	Spam Caught	Ham Blocked	Accuracy
dot	1.0	72.89%	0.00%	96.46%
dot	0.0	70.53%	0.00%	96.15 %
radial	1.0	11.79%	0.00%	88.49%
polynomial	1.0	28.88%	0.00%	90.72%
anova	1.0	73.87%	0.15%	96.46%
epachnenikov	1.0	11.79%	0.00%	88.49%

As shown in Table 5, all of these methods resulted in lower accuracy, with SVM showing a small improvement in the proper classification of legitimate messages.

It was only when I applied PCA for dimensionality reduction that Neural Network became a true contender. PCA greatly improved classification of both spam and legitimate messages, leaving it just short of Deep Learning in both measures. The use of PCA came at a cost, however: in my testing it took PCA over 250 ms to apply its model to each message. This delay makes PCA unfeasible for a time-sensitive setup like text messages—doubly so if the processing takes place on a mobile device. (Once PCA or another feature reduction took place, Neural Network proved quick.)

5.4 SVM

SVM gives intriguing results because, across almost every run, it perfectly identifies legitimate messages. This perfection likely would not translate into real life, but, as we shall see in the next section, an algorithm that identifies true messages with extreme accuracy is very useful even if its total accuracy is not as high as other algorithms.

SVM in RapidMiner has two main variables: the kernel and the tolerance for mis-classification. [3] The default kernel (dot) and a small tolerance for mis-classification proved the best combination. Table 6 has a selection of the SVM optimizations.

6 Recommendations

Based on the results shown in Table 2, there is no single algorithm that accomplishes the three goals of this study. Put briefly, those goals are:

1) Block as much spam as possible
2) Block few (ideally, no) legitimate messages
3) Apply a model quickly

Neural network performs well overall, though trailing Deep Learning in both spam detection and avoiding the misclassification of legitimate messages. However, the time it takes to apply PCA to a test message to performan dimensionality reduction (>250 ms) is unacceptable in a time-critical environment like text messaging.

Deep Learning is the best at blocking spam, but it does block a small percentage of true messages (though less than 1 in 800). SVM, while far behind in spam detection (73% vs. 90%), is very, very good at identifying legitimate messages. In addition, both algorithms, run quickly, allowing them to be run without unduly delaying a text message; in addition,

their short overall runtimes mean they can support larger training sets and/or frequent model updates). Both of these algorithms can play a role in a comprehensive spam filtering method:

- SVM is best when the sender is known (in the user's address book, perhaps, or known to have replied to a previous message from this sender). In this scenario, where the sender can be assumed to be legitimate, SVM's ability to block very few true messages is preferable to the best spam filtering. SVM, though, works well enough at catching spam (¿70%) to provide some protection if a phone or phone number is hacked.
- Deep Learning is best for unknown or suspicious text message senders. Deep Learning provides the best spam filtering (¿90%) with minimal blocking of legitimate messages (0.12%), making it ideal for this scenario.

7 Conclusion

Text messages, both in traditional SMS form as well as newer instant messaging apps, are a field ripe for spam. The short length of these messages can make gleaning enough information to filter spam messages difficult.

Machine learning, however, provides a means of discovering spam messages with good results. Deep Learning and SVM, in particular, provide two useful choices: high absolute accuracy (Deep Learning) or extremely low instances of blocking legitimate messages (SVM). Used in combination, these two algorithms provide robust spam protection.

References

[1] Almeida, Tiago A., Jos Mara G. Hidalgo, and Akebo Yamakami. "Contributions to the study of SMS spam filtering: new collection and results." Proceedings of the 11th ACM symposium on Document engineering. ACM, 2011.
[2] Almeida, Tiago A, and Jos Mara Gmez Hidalgo. SMS Spam Collection v. 1. SMS Spam Collection, www.dt.fee.unicamp.br/˜tiago/smsspamcollection/.
[3] RapidMiner Studio Free Edition. (2018). RapidMiner GmbH.
[4] https://community.rapidminer.com/discussion/34303/text-analysis-on-documents-collection-coming-from-a-csv.
[5] https://stackoverflow.com/questions/15878053/how-to-test-on-testset-using-rapidminer
[6] https://community.rapidminer.com/discussion/31723/text-mining-and-the-word-list
[7] https://www.quora.com/What-are-the-best-machine-learning-techniques-for-text-classification